USENIX Security '24 Artifact Appendix: CO3 Concolic Co-execution for Firmware

Changming Liu
Northeastern University

Alejandro Mera
Northeastern University

A Artifact Appendix

A.1 Abstract

This artifact contains all the source code and scripts developed for the CO3 project. It has three main components: 1. the instrumentation pass, 2. the firmware source code, 3. the orchestrator.

The workflow is that, we use the instrumentation pass to instrument the application code inside the firmware source code. The instrumented firmware, when flashed into the microcontroller board, will report the program status either through the USB port, or a serial port to the workstation. The orchestrator running on the workstation will receive the information and then start doing concolic execution.

The difficulty in reproducing this experiment is the lack of the physical MCU that runs the instrumented firmware. Moreover, flashing the firmware to the board requires some domain knowledge and is unfriendly to people who do not have experience with the MCU.

As a result, we prepare a Linux program that can be instrumented and carry out the same process mentioned above. This experiment is not mentioned in our main paper, its sole purpose is to carry out the experiment in a more controlled and accessible environment (i.e., the linux OS) compared to the original MCU board environment.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None

A.2.2 How to access

The whole code can be accessible at git clone https://github.com/Lawliar/CO3/releases/tag/v1.0.0

Please clone this repo and follows the readme.

A.2.3 Hardware dependencies

The original CO3 requires the MCU. However, since purchasing a MCU and the subsequent flashing the firmware are not feasible, we prepare a linux program as a replacement to perform concolic co-execution.

A.2.4 Software dependencies

- Docker

A.2.5 Benchmarks

CGC-CROMU-00001

A.3 Set-up

We have packaged all the building process into the dockerfile. By running

```
docker build . -t co3
```

one will be able to set up the environment, build the instrumentation pass, instrument the CGC-CROMU-00001 program, build the orchestrator. These are the same components used in the firmware setting, with special configuration to run on the workstation.

A.3.1 Installation

Everything is inside the docker container once it is built. Then run

```
docker run -it co3 /bin/bash
```

to launch the container.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *The core design of CO3 is to instrument the target program. The instrumented program will automatically report symbolic state to the orchestrator. The orchestrator will receive the messages from the instrumented program and generate alternative inputs through concolic execution.*

A.4.2 Experiments

(E1): Instrumentation and generate Symbolic Value Flow Graph (SVFG) [a few seconds computer time]: Using the LLVM pass that we develop, one should be able to

instrument a program. After instrumentation, we generate a binary that has reporting logic embedded in it. Besides, the SVFG should also be produced.
How to: This instrumentation process was automated in `/CO3_SOURCE/sym_runtime/CROMU_00001/CMakeLists.txt`. This process has already been executed when building the docker image.
However, this process can be re-executed. One can simply delete the `build` and `intermediate_results` folder, and run cmake to generate them again.
Execution: Run `cmake` following the command in dockerfile. Specifically,

- create a `build` folder under `CROMU_00001`
- Under the `build` folder, run `cmake -DCO3_32BIT=OFF -DCO3_NO_SHADOW=OFF -DCO3_DOCKER_BUILD=ON ..`
- `make`

Results:
- The SVFGs will be generated under `intermediate_results`.
- The executable will be generated under `build`.
- Both of them will be in the same level.

(E2): Concolic Co-execution. The instrumented program will communicate with the orchestrator to perform concolic execution.
How to: We launch the instrumented program, it then will wait for input from the orchestrator. We then launch the orchestrator, specifying the SVFG folder. Since this process requires running two processes at the same time, we wrote a script to simplify the process.
Execution: One can simply go to `/CO3_SOURCE/utils`, then execute

```
python3 co3_workstation.py -p 0 -b 0
```

Results: One can expect to see concolic execution be performed as the output shows. Feel free to execute as long as you feel like, and terminate with `ctrl + c`. The generated inputs will be in `/CO3_SOURCE/sym_runtime/CROMU_00001/intermediate_results/output`.
Possible Issue: If the python script complains about the missing concreteInputs.bin. It means that, it tries to read the seed input from the `/CO3_SOURCE/sym_runtime/CROMU_00001/intermediate_results/concreteInputs.bin` but it was missing. One can simply copy the PoV file `/CO3_SOURCE/sym_runtime/CROMU_00001/pov/first1K.xml.bin2` to `/CO3_SOURCE/sym_runtime/CROMU_00001/intermediate_results/concreteInputs.bin`, and run `touch /CO3_SOURCE/sym_runtime/CROMU_00001/intermediate_results/fileUSB.bin`

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2024/`.